Processamento de Linguagens e Compiladores (3 ano de Ciências)

Trabalho Prático 2

Relatório de Desenvolvimento

João Silva
(a87939)

João Barbosa
(a82044)

23 de janeiro de 2021

Resumo

Neste trabalho criamos uma linguagem imperativa simples, no qual implementamos algumas funções básicas de outras liguagens, por exemplo declarações de variaveis inteiras, ciclos for, if statements, entre outros. Com este trabalho pretendemos criar uma linguagem de programação, na qual, através da liguangem flex e yacc criamos processador de linguagem que origina um compilador,sendo que este gera código para uma máquina de stack virtual (VM no nosso caso).

Conteúdo

Capítulo 1

Introdução

No âmbito da cadeira de Processamento de linguagens e compiladores realizamos este trabalho prático com o objetivo de colocar em prática os conhecimentos teóricos adquiridos ao longo das aulas. Este trabalho prático visa aumentar a capacidade de trabalhar em engenharia de linguagens e em programação generativa. Para além disso, também serve para desenvolver processadores de linguagem, um compilador e utilizar geradores de compiladores. Com este trabalho iremos aumentar a capacidade de utilização de ferramentas de apoio à programação, ampliar as potencialidades de uso do Linux e da linguagem imperativa C.

1.1 Um compilador gerador de código para uma máquina virtual

Área: Processamento de Linguagens e compiladores

Enquadramento criar uma liguagem de programação imperativa simples com base numa já existente.

Contexto , tornar a leitura da linguagem criada uma tarefa mais fácil.

Problema desenvolver um processador de linguagem/compilador gerador de código para uma máquina de stack virtual (VM).

Objetivo facilitar a leitura e escrita da linguagem criada.

Resultados ou Contributos Neste trabalho conseguimos realizar todos os pontos pedidos em linguagem flex/yacc, menos o quarto ponto devido a problemas com o mod.

Estrutura do documento Neste trabalho será abordado as formas como resolvemos o processamento da linguagem, assim como a geração de código para a máquina virtual.

Estrutura do Relatório

No capitulo 1 fazemos uma breve introdução ao trabalho, referindo o seu enquadramento, estrutura, contexto, objetivos, problema e os resultados a obter. No capítulo 2 falamos sobre o problema e a sua descrição e o que temos de fazer para cada exemplo do trabalho, os dados fornecidos, os pedidos e as relações entre as alineas. No capitulo 3 é referido qual foi a nossa estratégia para a resolução deste trabalho, dizendo as estruturas de dados e algoritmos utilizados. No capitulo 4 são exibidos os resultados para os pontos pedidos. No capítulo 5 retiramos as conclusões, referimos formas diferentes de abordagem do problema e alguns problemas da implementação do código. No capítulo 6 apresentamos a totalidade do nosso código em flex e em yacc.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

O principal objetivo é criar com base numa liguagem imperativa já existente, uma liguagem imperativa simples para tornar a leitura e escrita do utilizador numa tarefa menos custosa e mais fácil. Com este trabalho pretendemos desenvolver um compilador gerador de código para uma máquina de stack virtual, no qual, através da liguangem flex e yacc criamos um processador de linguagem e um compilador.

2.2 Especificação do Requisitos

2.2.1 Dados

Foi-nos fornecido um ficheiro que contem as instruçoes para utilização da máquina virtual (VM).

2.2.2 Pedidos

No primeiro ponto é nos pedido que seja possível declarar variáveis atómicas do tipo inteiro, com as quais se podem efetuar as habituais operações aritméticas e relacionais. No segundo ponto é nos pedido que seja possível atribuir valores de expressões numéricas a variáveis atómicas. No terceiro ponto é nos pedido que consigamos escrever no standard input e escrever no standard output. No quarto ponto é nos pedido que seja implementado um ciclo FOR-DO. No quinto ponto é nos pedido que seja possivel declarar e manusear variáveis do tipo array (a 1 ou 2 dimensões).

2.2.3 Relações

Os vários pontos estão relacionados, pois só ao declarar a variável é que conseguimos aceder posteriormente ao valor armazenado na mesma, podendo ou não escrever este valor no standard input e depois armazenando esta variável num array atraves de um ciclo FOR-DO, por exemplo, existindo várias maneiras de interligar estes cinco pontos.

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de Dados

```
enum cat {Var, Array, Funct};
enum tipo {Int, Car, Arr};

struct my_struct {
    char name[10];             /* key (string is WITHIN the structure) */
    int end;
    enum cat category;
    enum tipo tipo;
    int dim;
    UT_hash_handle hh;         /* makes this structure hashable */
};
```

3.2 Algoritmos

```
HASH_ADD_STR(head,strfield,add) // que atribui uma variável a um nodo da hashmap com o seu devi
HASH_FIND_STR(head,findstr,out) // que procura uma variável na hashmap, esta funçao normalmente
char *errArr (char *id, char *expr, int dim, int *count) // que escreve para o ficheiro com as
```

Capítulo 4

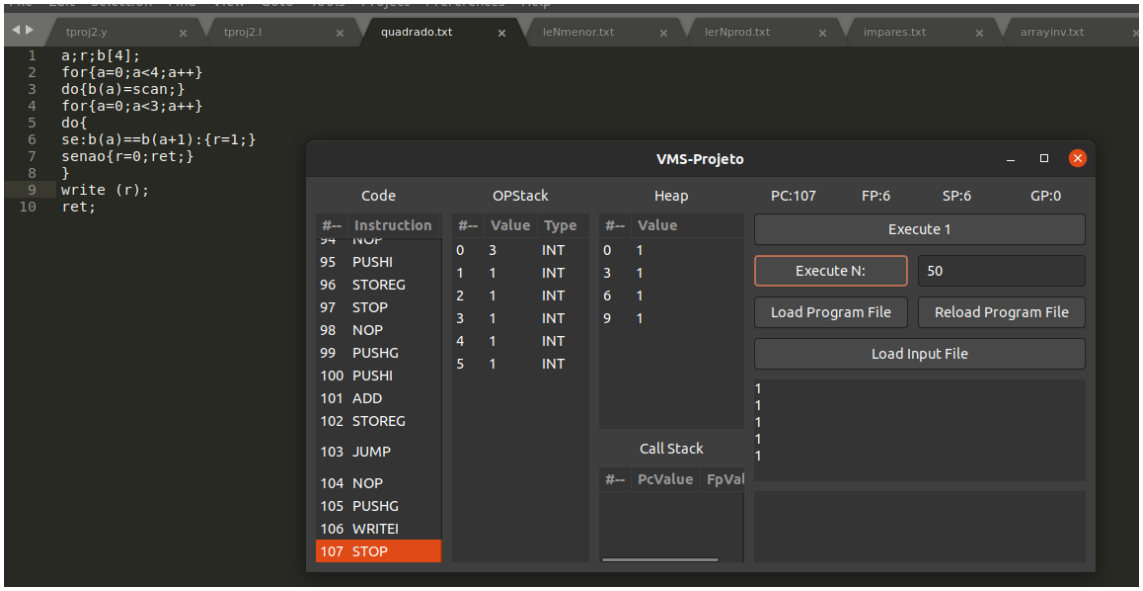
Codificação e Testes

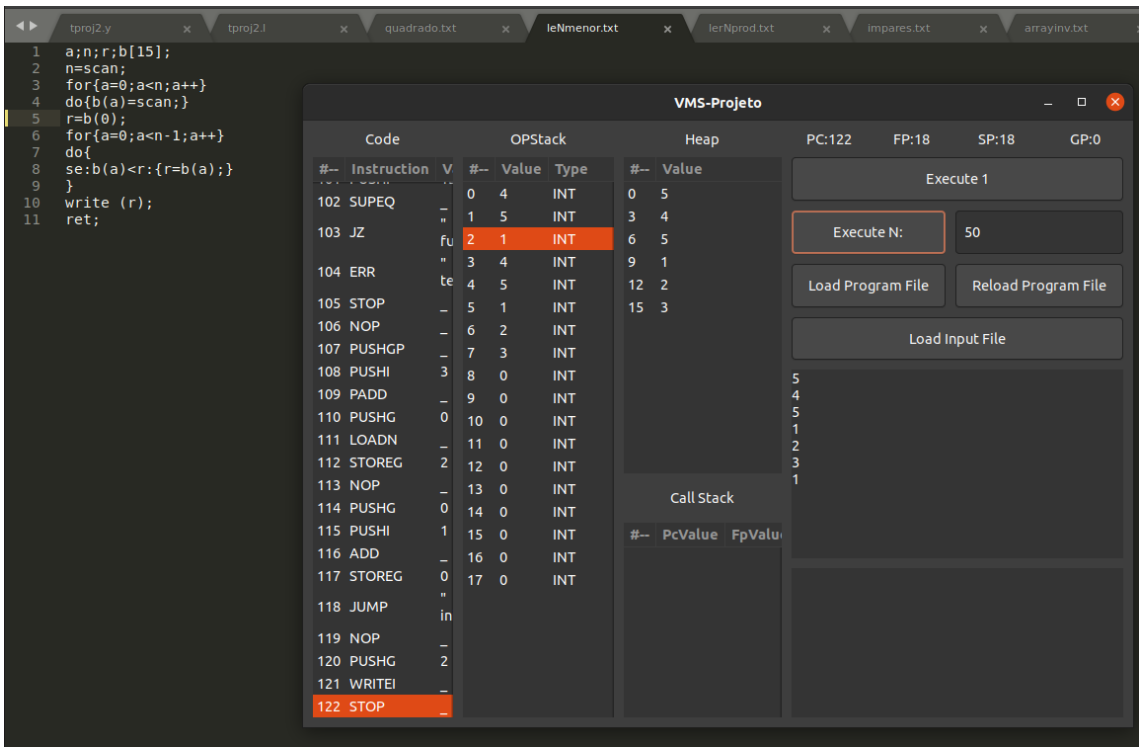
4.1 Alternativas, Decisões e Problemas de Implementação

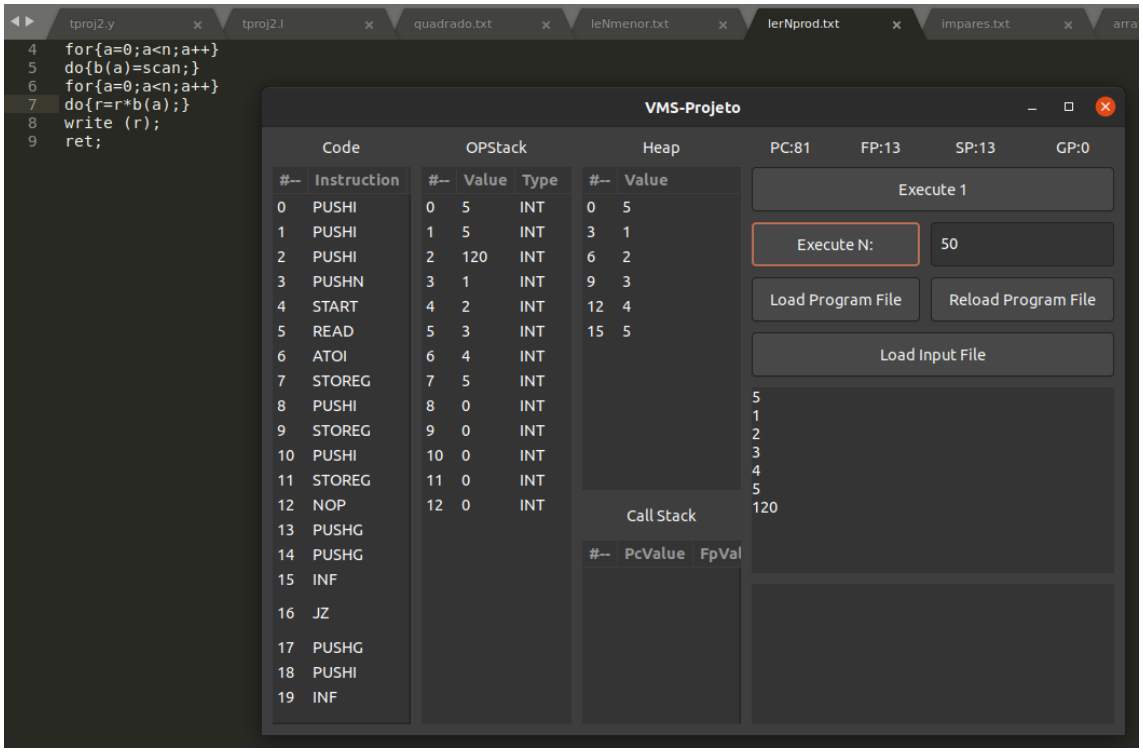
Em relação às estruturas de gestão das informações relativas a cada variável, escolhemos uma estrutura que nos fosse ser útil para este projeto, ou seja, uma estrutura que pudesse trabalhar como tabela de identificadores normalmente utilizada nos compiladores habituais. Na linguagem de "design de compiladores", uma tabela de identificadores ou tabela de símbolos, é a estrutura de dados usada por compiladores ou interpretadores, onde cada 'identificador' no código do programa está associado com a informação com a sua declaração e situações onde aparece na linguagem. Podemos dizer, em outras palavras, que cada entrada na tabela de símbolos guarda a informação relacionada com a entrada na tabela correspondente ao respetivo símbolo ou variável. Por isso, o mínimo de informação que pode existir na tabela é o nome da variável (ou símbolo) e a sua localização ou endereço na stack. Também como vantagem para o compilador, como também é o normal para linguagens "high-level", em cada entrada da tabela, também deveria estar incluído o tipo, a categoria, a dimensão de cada variável. Apesar de toda esta informação não ser diretamente importante para os comandos assembly, ajuda na correção de situações de mau uso das variáveis, como por exemplo, em tentativas de acessos a posições inexistentes dos arrays ou tentativas de acesso a variáveis que não existem, ou seja, que não foram inicializadas. Também consideramos a possibilidade de mostrar quais as variáveis existentes na estrutura gerada pelo código, mas isto a funcionar como extra, não influenciando a tabela nem os comandos assembly gerados. Entre todas as estruturas de dados normalmente conhecidas, havia muitas opções viáveis para este projeto, pois toda a gente nesta área não se cinge pela mesma estrutura de dados, indicando assim que não existe uma estrutura errada ou certa para este problema. Entre as estruturas normalmente utilizadas em compiladores, seria muito normal ser trabalhado com AVL-trees, listas ligadas, hash tables (não se considerou arrays pois são muito menos eficientes comparadas às outras estruturas em relação à procura de um certo elemento e às possibilidades de indexação de cada elemento). Para o nosso caso, como esta linguagem supostamente não vai lidar com várias variáveis ao mesmo tempo, decidimos utilizar a hash table porque é uma estrutura de dados com uma procura eficiente de certo elemento, não compromete a eficiência do compilador e torna muito fácil a indexação da entrada de cada variável na tabela, revelando-se assim uma estruturas de dados boa para tratar deste problema em mãos. De novo, reforçamos outra vez a ideia que existem múltiplas estruturas de dados boas para trabalhar em compiladores, para realizar este projeto. Simplesmente escolhemos a hash table por conveniência.

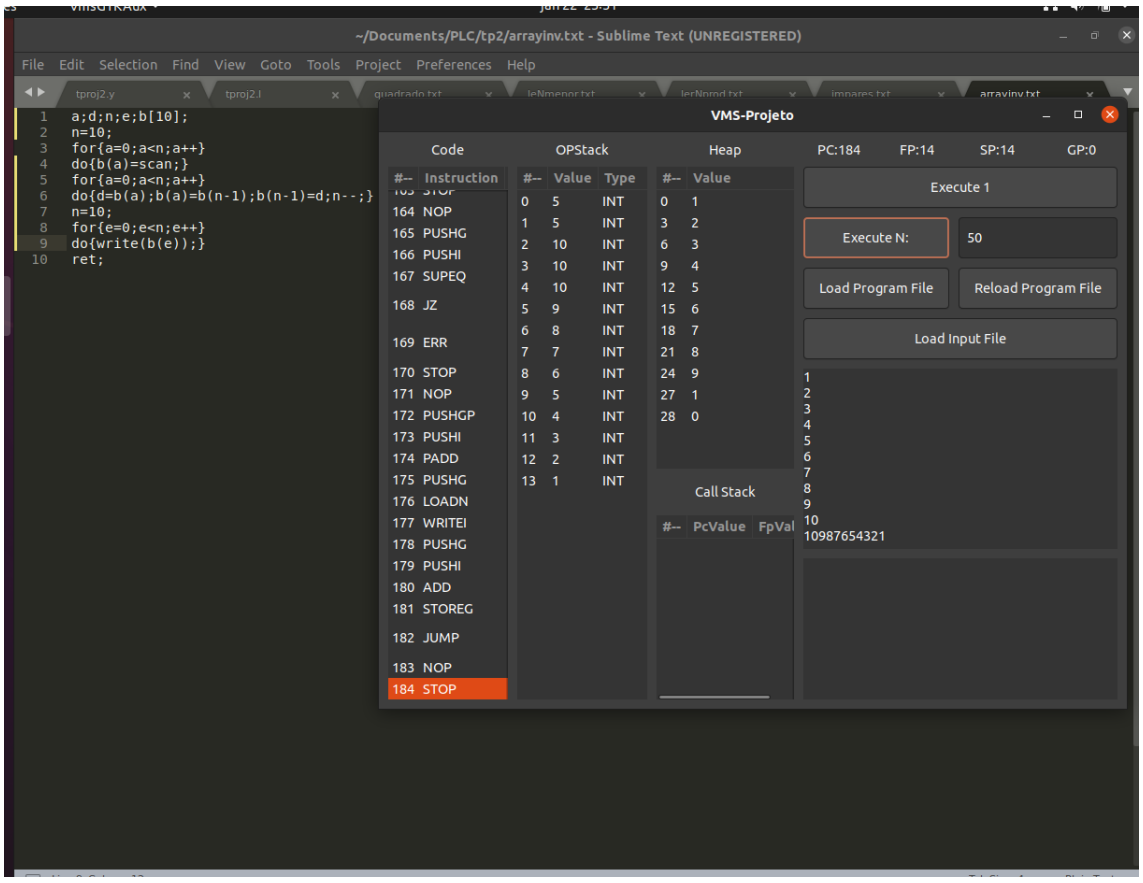
4.2 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos com alguns ficheiros criados por nós de modo a testar se os respectivos resultados obtidos correspondiam com o que seria o correto para cada caso:






tproj2.y
tproj2.l
quadrado.txt
leNmenor.txt
lerNprod.txt
impares.txt
4 for{a=0;a<n;a++}
5 do{b(a)=scan;}
6 for{a=0;a<n;a++}
7 do{r=r*b(a);}
8 write (r);
9 ret;
VMS-Projeto
Code
OPStack
Heap
PC:81
FP:13
SP:13
GP:0
#-- Instruction
0 PUSHI
1 PUSHI
2 PUSHI
3 PUSHN
4 START
5 READ
6 ATOI
7 STOREG
8 PUSHI
9 STOREG
10 PUSHI
11 STOREG
12 NOP
13 PUSHG
14 PUSHG
15 INF
16 JZ
17 PUSHG
18 PUSHI
19 INF
#-- Value Type
0 5 INT
1 5 INT
2 120 INT
3 1 INT
4 2 INT
5 3 INT
6 4 INT
7 5 INT
8 0 INT
9 0 INT
10 0 INT
11 0 INT
12 0 INT
#-- Value
0 5
3 1
6 2
9 3
12 4
15 5
Call Stack
#-- PcValue FpVal
Execute 1
Execute N:
50
Load Program File
Reload Program File
Load Input File
5
1
2
3
4
5
120


~/Documents/PLC/tp2/arrayinv.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
tproj2.y
tproj2.l
arrayinv.txt
1 a;d;n;e;b[10];
2 n=10;
3 for{a=0;a<n;a++}
4 do{b(a)=scan;}
5 for{a=0;a<n;a++}
6 do{d=b(a);b(a)=b(n-1);b(n-1)=d;n--;}
7 n=10;
8 for{e=0;e<n;e++}
9 do{write(b(e));}
10 ret;
VMS-Projeto
Code
OPStack
Heap
PC:184
FP:14
SP:14
GP:0
#-- Instruction
164 NOP
165 PUSHG
166 PUSHI
167 SUPEQ
168 JZ
169 ERR
170 STOP
171 NOP
172 PUSHGP
173 PUSHI
174 PADD
175 PUSHG
176 LOADN
177 WRITEI
178 PUSHG
179 PUSHI
180 ADD
181 STOREG
182 JUMP
183 NOP
184 STOP
#-- Value Type
0 5 INT
1 5 INT
2 10 INT
3 10 INT
4 10 INT
5 9 INT
6 8 INT
7 7 INT
8 6 INT
9 5 INT
10 4 INT
11 3 INT
12 2 INT
13 1 INT
#-- Value
0 1
3 2
6 3
9 4
12 5
15 6
18 7
21 8
24 9
27 1
28 0
Call Stack
#-- PcValue FpVal
Execute 1
Execute N:
50
Load Program File
Reload Program File
Load Input File
1
2
3
4
5
6
7
8
9
10
10987654321

Capítulo 5

Conclusão

Ao longo da realização deste trabalho deparamo-nos com algumas dificuldades, mas com esforço e dedicação, bem como com a ajuda do professor conseguimos concluir o projeto solicitado. Com este trabalho aumentamos a capacidade de trabalhar com gramáticas independentes de contexto. Paralelamente a realização desta atividade foi importante para desenvolver o hábito de escrever a documentação em LaTeX. Por fim, gostaríamos de agradecer ao professor Pedro Henriques a colaboração prestada e toda a disponibilidade evidenciada para a resolução das dificuldades com que nos íamos deparando.

Capítulo 6

Código do Programa

Apresentamos a seguir o código do programa que foi desenvolvido.
CÓDIGO FLEX

```
%option noyywrap

%%
[+\-*/=;:()\[\]\{\}|\r]    {return (yytext[0]);}
"=="                {return (IG);}
"~="                {return (DIF);}
">="                {return (IMA);}
">"                 {return (MA);}
"<="                {return (IME);}
"<"                 {return (ME);}
"++"                {return (INC);}
"--"                {return (DEC);}
(?i:scan)           {return (SC);}
(?i:for)            {return (FOR);}
(?i:se)             {return (SE);}
(?i:ret)            {return (RET);}
(?i:senao)          {return (SENAO);}
(?i:do)             {return (DO);}
(?i:true)           {return (TRUE);}
(?i:false)          {return (FALSE);}
(?i:mod)            {return (MOD);}
(?i:write)          {return (WRITE);}

[A-Za-z]+           {yylval.valC= strdup(yytext);return (ID);}

[0-9]+   {yylval.valN = atoi(yytext); return (NUM);}
.|\n                {;}
%%
```

CÓDIGO YACC

```
  %{
```

```
#include <stdlib.h>  /* malloc */
#include <stdio.h>
#include <string.h>
#include "uthash.h"

enum cat {Var, Array, Funct};
enum tipo {Int, Car, Arr};

struct my_struct {
    char name[10];             /* key (string is WITHIN the structure) */
    int end;
    enum cat category;
    enum tipo tipo;
    int dim;
    UT_hash_handle hh;         /* makes this structure hashable */
};


int yylex();
void yyerror(char *s);
char *errArr (char *id, char *expr, int dim, int *count);
float TabId[26];
FILE *exe;
struct my_struct *s, *tmp, *users = NULL;

int cfor = 1;
int cse = 1;
int carr = 1;
int fp = 0;

%}

%union{ int valN; char *valC;char *inst;}
%token <valN>NUM
%token <valC>ID
%token WRITE
%token FOR DO
%token SE SENAO
%token SC RET
%token TRUE FALSE
%token MOD IG DIF IMA MA IME ME INC DEC

%type <inst> Inics Comms Prints
%type <inst> Atrib Ciclo Opcao
%type <inst> Iter Conds Expr Termo Fator

%start Codigo
```

```
%%

Codigo : Inics Comms RET ';' {fprintf (exe, "%sstart\n%sstop",$1,$2);}

Inics : Inics ID ';'                    { s = (struct my_struct *)malloc(sizeof *s);
                                          strcpy(s->name, $2);
                                          s->end = fp;
                                          s->category = Var;
                                          s->tipo = Int;
                                          s->dim = 1;
                                          HASH_ADD_STR( users, name, s );
                                          fp++;
                                          asprintf(&$$,"%s\tpushi 0\n",$1);
                                        }
      | Inics ID '[' NUM ']' ';'        { // opção para declarar um array
                                          s = (struct my_struct *)malloc(sizeof *s);
                                          strcpy(s->name, $2);
                                          s->end = fp;
                                          s->category = Array;
                                          s->tipo = Arr;
                                          s->dim = $4;
                                          fp= fp + s->dim;
                                          HASH_ADD_STR( users, name, s );
                                          asprintf(&$$,"%s\tpushn %d\n",$1,$4);
                                        }
      | ID ';'                          { s = (struct my_struct *)malloc(sizeof *s);
                                          strcpy(s->name, $1);
                                          s->end = fp;
                                          s->category = Var;
                                          s->tipo = Int;
                                          s->dim = 1;
                                          HASH_ADD_STR( users, name, s );
                                          fp++;
                                          asprintf(&$$,"\tpushi 0\n");
                                        }
      | ID '[' NUM ']' ';'              { // opção para declarar um array
                                          s = (struct my_struct *)malloc(sizeof *s);
                                          strcpy(s->name, $1);
                                          s->end = fp;
                                          s->category = Array;
                                          s->tipo = Arr;
                                          s->dim = $3;
                                          fp= fp + s->dim;
                                          HASH_ADD_STR( users, name, s );
                                          asprintf(&$$,"\tpushn %d\n",$3);
                                        }
      ;
```

```
Comms : Comms Atrib ';'              {asprintf(&$$,"%s%s",$1,$2);}
      | Comms Ciclo                  {asprintf(&$$,"%s%s",$1,$2);}
      | Comms Opcao                  {asprintf(&$$,"%s%s",$1,$2);}
      | Comms Iter ';'               {asprintf(&$$,"%s%s",$1,$2);}
      | Comms RET ';'                {asprintf(&$$,"%s\tstop\n",$1);}
      | Comms Prints ';'             {asprintf(&$$,"%s%s",$1,$2);}
      | Atrib ';'                    {asprintf(&$$,"%s",$1);}
      | Ciclo                        {asprintf(&$$,"%s",$1);}
      | Opcao                        {asprintf(&$$,"%s",$1);}
      | Iter ';'                     {asprintf(&$$,"%s",$1);}
      | Prints ';'                   {asprintf(&$$,"%s",$1);}
      | RET ';'                      {asprintf(&$$,"%s\tstop\n");}
      ;

Prints: WRITE '(' Expr ')'           {asprintf(&$$,"%s\twritei\n",$3);}
      ;

Ciclo : FOR '{' Atrib ';' Conds ';' Iter '}' DO '{' Comms '}'     { asprintf(&$$,"%siniciofor%d
      ;

Opcao : SE ':' Conds ':' '{' Comms '}' SENAO '{' Comms '}'              { asprintf(&$$,"%s\tjz
      | SE ':' Conds ':' '{' Comms '}'                                  { asprintf(&$$,"%s\tjz
      ;

Iter : ID INC                    { HASH_FIND_STR( users, $1, s);
                                     if (s) {
                                       if ((s->category == Var) && (s->tipo == Int)) {
                                       asprintf(&$$,"\tpushg %d\n\tpushi 1\n\tadd\n\tstoreg %d\n
                                       }
                                       else { printf("\n\n   Erro Desconhecido   \n\n"); }
                                     }
                                     else { printf("\n\n Erro : foi tentado utilizar a variável
                                   }
     | ID DEC                    { HASH_FIND_STR( users, $1, s);
                                    if (s) {
                                       if ((s->category == Var) && (s->tipo == Int)) {
                                       asprintf(&$$,"\tpushg %d\n\tpushi 1\n\tsub\n\tstoreg %d\n
                                       }
                                       else { printf("\n\n   Erro Desconhecido   \n\n"); }
                                     }
                                     else { printf("\n\n Erro : foi tentado utilizar a variável
                                   }
     | INC ID                     { HASH_FIND_STR( users, $2, s);
                                     if (s) {
                                       if ((s->category == Var) && (s->tipo == Int)) {
                                       asprintf(&$$,"\tpushg %d\n\tpushi 1\n\tadd\n\tstoreg %d\n
                                       }
                                       else { printf("\n\n   Erro Desconhecido   \n\n"); }
```

```
                                    }
                                    else { printf("\n\n Erro : foi tentado utilizar a variável
                                  }
      | DEC ID                   { HASH_FIND_STR( users, $2, s);
                                    if (s) {
                                      if ((s->category == Var) && (s->tipo == Int)) {
                                      asprintf(&$$,"\tpushg %d\n\tpushi 1\n\tsub\n\tstoreg %d\n
                                      }
                                      else { printf("\n\n   Erro Desconhecido   \n\n"); }
                                    }
                                    else { printf("\n\n Erro : foi tentado utilizar a variável
                                  }
      ;

Conds : Expr IG Expr          { asprintf(&$$,"%s%s\tequal\n", $1, $3); }
      | Expr DIF Expr         { asprintf(&$$,"%s%s\tinf\n%s%s\tsup\n\tadd\n", $1, $3,$1,$3); }
      | Expr IME Expr         { asprintf(&$$,"%s%s\tinfeq\n", $1, $3); }
      | Expr IMA Expr         { asprintf(&$$,"%s%s\tsupeq\n", $1, $3); }
      | Expr MA Expr          { asprintf(&$$,"%s%s\tsup\n", $1, $3); }
      | Expr ME Expr          { asprintf(&$$,"%s%s\tinf\n", $1, $3); }
      ;

Atrib : ID '=' Expr     {  HASH_FIND_STR(users, $1, s);
                           if (s) {
                           if ((s->category == Var) && (s->tipo == Int)) {
                              asprintf(&$$,"%s\tstoreg %d\n", $3, s->end);
                              }
                              else { printf("\n\n   Erro Desconhecido 4  \n\n"); }
                           }
                           else { printf("\n\n Erro : foi tentado utilizar a variável %s quando e
                        }
      | ID '=' SC    { HASH_FIND_STR(users, $1, s);
                           if (s) {
                           if ((s->category == Var) && (s->tipo == Int)) {
                              asprintf(&$$,"\tread\n\tatoi\n\tstoreg %d\n",s->end);
                              }
                              else { printf("\n\n   Erro Desconhecido 5  \n\n"); }
                           }
                           else { printf("\n\n Erro : foi tentado utilizar a variável %s quando e
                         }
      | ID '(' Expr ')' '=' Expr  {  // opção para alterar a uma certa posição do array
                                    HASH_FIND_STR( users, $1, s);
                                    if (s) {
                                      if ((s->category == Array) && (s->tipo == Arr)) {
                                        char *error_str = errArr($1, $3, s->dim, &carr);
                                        asprintf(&$$,"%s\tpushgp\n\tpushi %d\n\tpadd\n%s%s\tstore
                                      }
                                      else { printf("\n\n   Erro Desconhecido 1  \n\n"); }
```

```
                                    }
                                    else { printf("\n\n Erro : foi tentado utilizar a variável %s
                                  }
      | ID '(' Expr ')' '=' SC     { HASH_FIND_STR( users, $1, s);
                                    if (s) {
                                      if ((s->category == Array) && (s->tipo == Arr)) {
                                        char *error_str = errArr($1, $3, s->dim, &carr);
                                        asprintf(&$$,"%s\tpushgp\n\tpushi %d\n\tpadd\n%s\tread\n\
                                      }
                                      else { printf("\n\n   Erro Desconhecido 3  \n\n"); }
                                    }
                                    else { printf("\n\n Erro : foi tentado utilizar a variável %s
                                   }
      ;


Expr  : Termo              { asprintf(&$$,"%s", $1); }
      | Expr '+' Termo     { asprintf(&$$,"%s%s\tadd\n", $1, $3);}
      | Expr '-' Termo     { asprintf(&$$,"%s%s\tsub\n", $1, $3);}
      ;

Termo : Fator               { asprintf(&$$,"%s", $1);}
      | Termo '*' Fator     { asprintf(&$$,"%s%s\tmul\n", $1, $3);}
      | Termo '/' Fator     { asprintf(&$$,"%s%s\tdiv\n", $1, $3);}
      | Termo MOD Fator     { asprintf(&$$,"%s%s\tmod\n", $3, $1);}
      ;

Fator : NUM              { asprintf(&$$,"\tpushi %d\n", $1);}
      | '-' NUM          { asprintf(&$$,"\tpushi -%d\n", $2);}
      | TRUE             { asprintf(&$$,"\tpushi 1\n");}
      | FALSE            { asprintf(&$$,"\tpushi 0\n"); }
      | '(' Expr ')'     { asprintf(&$$,"\tpushi %d\n", $2);}
      | ID                          { HASH_FIND_STR( users, $1, s);
                                      if (s) {
                                            if ((s->category == Var) && (s->tipo == Int)) {
                                                  asprintf(&$$,"\tpushg %d\n", s->end);
                                            } else { printf("\n\n   Erro Desconhecido   \n\n");
                                      } else { printf("\n\n Erro : foi tentado utilizar a variá
                                    }
      | ID '(' Expr ')'             {  HASH_FIND_STR( users, $1, s);
                                       if (s) {
                                         if ((s->category == Array) && (s->tipo == Arr)) {
                                         char *error_str = errArr($1, $3, s->dim, &carr);
                                         asprintf(&$$,"%s\tpushgp\n\tpushi %d\n\tpadd\n%s\tloadn
                                       }
                                       else { printf("\n\n   Erro Desconhecido 1  \n\n"); }
                                     }
                                     else { printf("\n\n Erro : foi tentado utilizar a variável
```

```
                                        }
      ;

%%

#include "lex.yy.c"
void yyerror (char* s){ printf("%s\n",s);}

char *errArr (char *id, char *expr, int dim, int *count) {
    char *r, *inferior, *greater, *error_str;
    asprintf (&greater, "%s\tpushi %d\n\tsupeq\n\tjz func%d\n\terr \"tentativa de acesso a posica
                expr, dim, *count, id, *count);
    asprintf (&inferior, "%s\tpushi 0\n\tinf\n\tjz func%d\n\terr \"tentativa de acesso a posicao
                expr, *count + 1, id,*count + 1);
    asprintf (&error_str, "%s%s", inferior, greater);
    *count = *count + 2;

    return error_str;
}

char *errInicArr (char *id, char *expr, int *count) {
    char *r, *inferior, *error_str;
    asprintf (&inferior, "%s\tpushi 0\n\tinf\n\tjz func%d\n\terr \"tentativa de acesso a posicao
                expr, *count + 1, id,*count + 1);
    asprintf (&error_str, "%s", inferior);
    *count = *count + 1;

    return error_str;
}

int main() {
      exe = fopen ("execute.vm", "w");
      char tp[20];
      yyparse();

      for(s=users; s != NULL; s=(struct my_struct*)(s->hh.next)) {
            printf("user id %d: name %s\n           end %d\n           category %d\n           t
      }

      /* free the hash table contents */
      HASH_ITER(hh, users, s, tmp) {
            printf("\n%s\n",s->name);
            HASH_DEL(users, s);
            free(s);
      }
      fclose (exe);
      return 0;
}
```